A Distance Measure for Large Graphs based on Prime Graphs

Sofiane Lagraa, Hamida Seba, Riadh Khennoufa, Abir Mbaya, Hamamache Kheddouci

A Distance Measure for Large Graphs based on Prime Graphs

Sofiane Lagraa[1]*, Hamida Seba[1]†, Riadh Khennoufa[2],
Abir M'Baya[1] and Hamamache Kheddouci[1]

[1] *Université de Lyon, CNRS*
Université Lyon 1, LIRIS, UMR5205, F-69622, France
[2] *Département INFO-Bourg, IUT Lyon1*
Université Lyon 1, F-01000, France

Abstract

Graphs are universal modeling tools. They are used to represent objects and their relationships in almost all domains: they are used to represent DNA, images, videos, social networks, XML documents, etc. When objects are represented by graphs, the problem of their comparison is a problem of comparing graphs. Comparing objects is a key task in our daily life. It is the core of a search engine, the backbone of a mining tool, etc. Nowadays, comparing objects faces the challenge of the large amount of data that this task must deal with. Moreover, when graphs are used to model these objects, it is known that graph comparison is very complex and computationally hard especially for large graphs. So, research on simplifying graph comparison gained in interest and several solutions are proposed. In this paper, we explore and evaluate a new solution for the comparison of large graphs. Our approach relies on a compact encoding of graphs called prime graphs. Prime graphs are smaller and simpler than the original ones but they retain the structure and properties of the encoded graphs. We propose to approximate the similarity between two graphs by comparing the corresponding prime graphs. Simulations results show that this approach is effective for large graphs.

Keywords: *Graph similarity, Graph decomposition, Quotient graph, Prime graph, Edit distance, Graph probing.*

*Sofiane Lagraa is actually at *TIMA, LIG, CNRS/Grenoble-INP/UJF*
†Corresponding author. Email: hamida.seba@univ-lyon1.fr

1 Introduction

It is well established that graphs are an effective and major way of representing objects in various domains and applications mainly those related to pattern recognition such as computer vision, data mining, biology, etc. A graph $G = (V, E)$ is a representation tool composed of a set of vertices V and a set of edges E with the cardinalities $|V(G)| = n$ and $|E(G)| = m$ where n is called the order of the graph and m its size. The set of edges E is a subset of $V \times V$ such that $(u, v) \in E$ means that vertices u and v are connected. Usually, a finite number of labels are associated with vertices and edges. So, in the graph representation, the vertices represent objects and the edges represent relations between these objects. The eventual labels represent objects' properties. When graphs are used to represent objects, the problem of comparing these objects turns into determining the similarity between the corresponding graphs.
Comparing graphs with a low computational cost and a high degree of precision remains a challenging issue despite years of investigations. Nowadays, with the big data challenge this topic is essential more than ever.
Graph comparison approaches are generally classified into two categories: exact approaches and inexact or fault-tolerant approaches. Exact approaches refer to the methods used to find out if two graphs are the same. The common related problems include graph isomorphism, sub-graph isomorphism and the maximum common subgraph [12, 16, 34, 58]. In these problems we look generally for an exact mapping between the vertices and edges of a query graph and the vertices and edges of a target graph.

Fault-tolerant graph comparison aims generally to compute a distance between the compared graphs. This distance measures how much these graphs are similar and help to deal with noise and distortion that are introduced during the process needed to model objects by graphs. The different stages of image encoding is perhaps the most illustrative example of such noise that graph comparison must deal with. Fault-tolerant graph comparison is also useful for search/rank based applications where a distance between the compared objects is needed. In some applications, graph similarity measures are intended to compute relatively suboptimal distances [16] that are compensated by a large reduction of the computational complexity of the comparison process. Several graph similarity measures have been proposed in the literature and several approaches have been used including genetic algorithms [29,56], neural networks [37], the theory of probability [15,41], clustering techniques [14,54], spectral methods [52,59], decision trees [35,36], etc. We refer the reader to [11, 12, 16, 19] for more exhaustive surveys. We focus here mainly on similarity measures that try to extend to graphs some of the properties defined in metric spaces.

Definition 1. *A metric space is an ordered pair (M, d) where M is a set and d is a metric on M, i.e., a function $d : M \times M \longrightarrow \mathbb{R}$ such that for any x, $y \in M$, the following holds:*

- $d(x, y) \geq 0$ *(non-negativity),*
- $d(x, y) = 0$ *iff* $x = y$ *(uniqueness),*

- $d(x, y) = d(y, x)$ *(symmetry) and*
- $d(x, z) \leq d(x, y) + d(y, z)$ *(triangle inequality).*

Perhaps, the most referenced metric is edit distance which defines the similarity of graphs by the minimum costing sequence of edit operations that convert one graph into the other [8, 53]. An edit operation is either an insertion, a suppression or a re-labeling of a vertex or an edge in the graph. A cost function associates a cost to each edit operation. Figure 1 shows an example of edit operations that are necessary to get the graph G_2 from G_1 with the suppression of two edges and a vertex and the relabeling of two vertices.

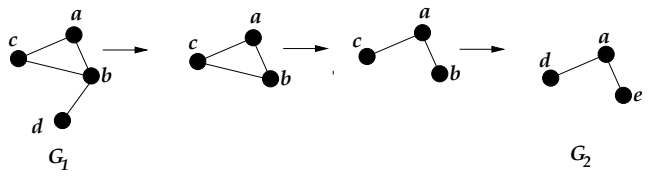


Figure 1: Example of edit operations.

Graph edit distance is a flexible graph similarity measure which is applicable to various kind of graphs [3, 8, 43, 51, 53]. It also defines a common theoretical framework that allows comparing different approaches of graph comparison. In fact, Bunke showed in [10] that under a particular cost function, graph edit distance computation is equivalent to the maximum common subgraph problem. In [7], the same author shows that the graph isomorphism and subgraph isomorphism problems can be reduced to graph edit distance. However, computing graph edit distance suffers from two main drawbacks:

1. A high computational complexity. The problem of computing graph edit distance is NP-hard in general [63]. The most known method for computing the exact value of graph edit distance is based on A^* [26] which is a best first search algorithm where the search space is organized as a tree. The root of the tree is the starting point of the algorithm. The internal vertices correspond to partial solutions and leaves represent complete solutions.
2. The difficulty related to defining cost functions [45].

The first drawback motivated several approximating solutions to compute graph edit distance. A comprehensive survey on graph edit distance and the approaches proposed to compute it can be found in [21]. We focus here on three independent but similar works that approximate the edit distance of two graphs by computing the edit distance between subgraphs of these graphs [47, 49, 63]. In [63], the authors introduce an novel method to compute an upper and lower bounds for the edit distance between two graphs in polynomial time. This method consists to use star representations of graphs and edit distance between stars defined as follows:

Definition 2. *[63] A star structure s is an attributed, single-level, rooted tree which can be represented by a 3-tuple $s = (r, \mathfrak{L}, \ell)$, where r is the root vertex, $\mathfrak{L}$ is the set of leaves and ℓ is a labeling function. Edges exist between r and any vertex in $\mathfrak{L}$ and no edge exists among vertices in $\mathfrak{L}$.*

Definition 3. *[63] Given two star structures s_1 and s_2, the edit distance between s_1 and s_2*

$$\lambda(s_1, s_2) = T(r_1, r_2) + d(\mathfrak{L}_1, \mathfrak{L}_2)$$

where

$$T(r_1, r_2) = \begin{cases} 0 & \text{if } \ell(r_1) = \ell(r_2), \\ 1 & \text{otherwise.} \end{cases}$$

$$d(\mathfrak{L}_1, \mathfrak{L}_2) = ||\mathfrak{L}_1| - |\mathfrak{L}_2|| + \mathfrak{M}(\mathfrak{L}_1, \mathfrak{L}_2)$$

$$\mathfrak{M}(\mathfrak{L}_1, \mathfrak{L}_2) = max\{|\Psi_{\mathfrak{L}_1}|, |\Psi_{\mathfrak{L}_2}|\} - |\Psi_{\mathfrak{L}_1} \cap \Psi_{\mathfrak{L}_2}|$$

$\Psi_{\mathfrak{L}}$ is the multiset of vertex labels in $\mathfrak{L}$.

The authors define the distance between two multisets of star structures. Subsequently, they define the mapping distance between two graphs based on the distance between their star representations.

Similarly to the work of [63], the authors of [49] define a mapping distance between two graphs based on the distance between their local structures. Each local structure contains a vertex and its incident edges (i.e., a star structure without the leaves). In [47], the authors describe a more general framework for this approach where the subgraphs are also stars. In this approach, the authors consider edge labels and the distance between sub-graphs may be different from edit-distance. This distance is given by the minimum-weight subgraph matching between the query and target graphs with respect to a cost function.

A relatively resembling distance, that does not use edit operations, is also defined in [28] where a different representation of the star structure is used. In this similarity measure, the star structure is called node signature and is represented by a vector containing the label of the vertex, its degree, and the set of labels of its incident edges. A distance between two node signatures is also defined and the distance between two graphs is then defined as an assignment problem in the matrix containing the distances between nodes signatures of the two compared graphs.

In [62], the authors present an extension of the similarity measure defined in [63] that ensures a better execution time. In this solution, the obtained subgraphs may be stars or bi-stars and are obtained by a graph coloring algorithm. However, this algorithm can be used only for trees.

In [64], the authors point-out the redundancy and the fixed-size of the substructures used in the aforementioned methods and propose a new approach that use variable-size non-overlapping substructures.

To overcome the second drawback and avoid the definition of edit costs, similarity measures

that do not use edit operations are also proposed. In [9], the authors propose a graph distance measure that is based on the maximal common subgraph of two graphs and prove that it is a metric, i.e., the measure satisfies the four properties of a usual metric namely: non-negativity, uniqueness, symmetry and triangle inequality. However, computing the maximal common subgraph of two graphs has a high computational complexity [9]. For this reason, Raymond et al. [48] propose a modified version of the measure defined in [9] where an initial screening process determines whether it is possible for the measure of similarity between the two graphs to exceed a minimum threshold for which it is acceptable to compute the maximum common subgraph. This screening process is based on computing graph invariants. Graph invariants have been efficiently used to solve the graph comparison problem in general and the graph isomorphism problem in particular. They are used for example in Nauty [34] which is one of the most efficient algorithm for graph and subgraph isomorphism testing. A vertex invariant, for example, is a number $i(v)$ assigned to a vertex v such that if there is an isomorphism that maps v to v' then $i(v) = i(v')$. Examples of invariants are the degree of a vertex, the number of cliques of size k that contain the vertex, the number of vertices at a given distance from the vertex, etc. Graph invariants are also the basis of graph probing [31] where a distance between two graphs is defined as the norm of their probes. Each graph probe is a vector of graph invariants.

In [60], the distance metric based on the maximum common subgraph defined in [9] is extended by a proposal to define the problem size with the union of the two compared graphs rather than the larger of the two graphs used in [9].

In [61], the authors show that we can evaluate graph distance with a high degree of precision by considering complex graph sub-structures in the distance. In fact, in some applications such as analysis of protein interaction graphs, some sub-structures of these graphs represent certain functional modules of cells or organisms. Hence, comparing these graphs in terms of substructure information is biologically meaningful [61]. The authors defined a new metric based on the concept of *Structure Abundance Vector*. Each element of a Structure Abundance Vector of a graph G contains the size of an occurrence of a predefined sub-structure in G. The *Structure Abundance Vector* is a generalization of the concept of graph invariants.
More recently, kernel based similarity measures are also proposed [5, 12, 22, 27, 42, 44]. The main idea is also to define similarity of graphs based on the similarity of substructures of these graphs.

In this work, we investigate a novel approach for comparing graphs. We propose a new distance measure to approximate the similarity between graphs. The key idea of our approach is simplifying the processing of large graphs by using a compact encoding of the graphs obtained by modular decomposition. Modular decomposition is a way of representing graphs that allows to encapsulate the contents of the graph into simpler graphs that are prime graphs. This means that they can not be further compacted by the use of modular decomposition. An earlier work that uses modular decomposition to compare graphs is described in [1]. However the approach does not use prime graphs. It focuses mainly on re-defining the concept of module or meaningful

substructure in the context of business process graphs. In the next section of this paper, we present the definition of modular decomposition and prime graphs. We also present the basic notations used throughout the paper. In Section 3, we define the prime graph based distance and describe how to compute it. Then, in Section 4, we analyze the proposed approach and show that this distance is a pseudo-metric. Section 5 investigates the application of this distance by performing comprehensive experimental studies over various datasets. Section 6 brings our remarks concluding the paper.

2 Preliminaries

In this section, we present the main concepts we use to define our similarity measure: quotient graphs, prime graphs and modular decomposition. We will use the notations summarized in Table 1 in the remaining of the paper.

Table 1: Notation

Symbol	Description
$G(V, E)$	undirected unlabeled graph with V its vertex set and E its edge set
$G = (V, E, f)$	undirected vertex labeled graph, f is a labeling function
$V(G)$	vertex set of the graph G
$E(G)$	edge set of the graph G
$\overline{G}$	the complement of the graph G
$deg(v)$	degree of vertex v
$\Delta(G)$	the greatest vertex degree in graph G
$G[X]$	the subgraph of G induced by the set of vertices X
$\mathcal{P}(G)$	prime graph of G
$\|T\|_1$	L_1-norm of vector T.

2.1 Modular Decomposition and Quotient Graphs

Graph decomposition refers generally to a kind of graph representations that aim to highlight regular or meaningful structures within the graph such as cycles, cliques, etc. Graph decompositions give a readable view of the structure of a graph and it can help to solve efficiently complex problems on graphs. Modular decomposition is one of the most known graph decompositions [24]. It was introduced by Gallai [20] to solve optimization problems. It was also used to recognize some graph classes [38, 39, 55]. For a survey of applications of modular decomposition see [18, 20, 38]. Modular decomposition generates a representation of a graph that highlights groups of vertices that have the same neighbors outside the group. These subsets of vertices are called modules.

Definition 4. *A module of a graph $G = (V, E)$ is a set $M \subseteq V$ of vertices where all vertices in M have the same neighbors in $V \setminus M$.*

The empty set, the singletons, and the vertex set $V(G)$ satisfy the definition of a module, they are called *trivial modules*. A graph that has only trivial modules is called a *prime graph*. Figure 2 illustrates an example of a graph and its modules.

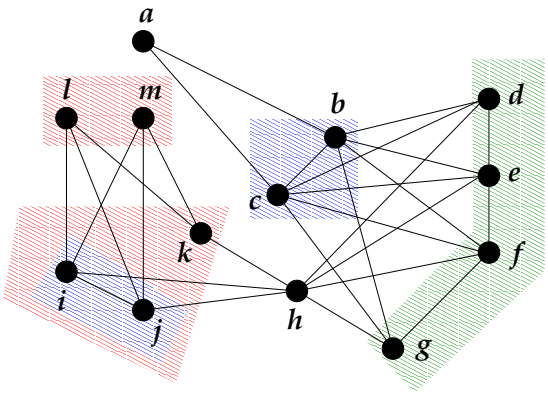


Figure 2: A graph and its modules.

We distinguish two kind of modules [18, 38]:

- *Weak module*: A module is weak if its overlaps another module. A module M_1 overlaps another module M_2 if $M_1 \cap M_2 \neq \varnothing$, $M_1 \setminus M_2 \neq \varnothing$ and $M_2 \setminus M_1 \neq \varnothing$.
- *Strong module*: A module is strong if it does not overlap any other module. A strong module can however contain other strong modules. A strong module that is not contained by any other module is a *maximal strong module*.

The graph of Figure 2 contains only strong modules as all its modules do not overlap.

Two strong modules M_1 and M_2 are either adjacent or non adjacent [38]. M_1 and M_2 are adjacent if every vertex of M_1 is adjacent to every vertex of M_2 and nonadjacent if no vertex of M_1 is adjacent to any vertex of M_2 [38].

In Figure 2, modules $\{c, b\}$ and $\{l, m\}$ are nonadjacent while modules $\{c, b\}$ and $\{d, e, f, g\}$ are adjacent.

We consider the partition P of the vertex set V of a graph $G = (V, E)$ where the elements of the partition are maximal strong modules. The relations of adjacency between the elements of the partition P, i.e., strong modules, constitute a new graph, the *quotient graph* G/P, whose vertices are the elements of the partition P. If P is a nontrivial modular partition, i.e. $P \neq V$ and $P \neq \{\{x\}, x \in V\}$, then G/P is a compact representation of all the edges that have endpoints in different partition classes of P [38]. For each partition class X in P, the subgraph $G[X]$ induced by X is called a factor and gives a representation of all edges with both endpoints in X. Therefore, the edges of G can be reconstructed given only the quotient graph G/P and its factors [38]. The term prime graph comes from the fact that a prime graph has only trivial quotients and factors [18].

The modular decomposition is a unique decomposition of the vertices of a graph into nested strong modules [18]. A strong module M of G can take one of the following types:

- **Prime:** Both, $G[M]$ and $\overline{G}[M]$ are connected.
- **Series:** All strong modules contained by M are adjacent to each other in $G[M]$. $\overline{G}[M]$ is not connected.
- **Parallel:** All strong modules contained by M are non-adjacent to each other in $G[M]$. $G[M]$ is not connected.

So, strong modules allow a compact encoding of a graph by replacing each module by a unique vertex. The quotient graph that results from recursively compacting all the strong modules is a prime graph. We will denotes such graph by $\mathcal{P}(G)$ for a given graph G. Figure 3 details in several steps how we get the final prime graph by compacting recursively the strong modules of the graph of Figure 2.

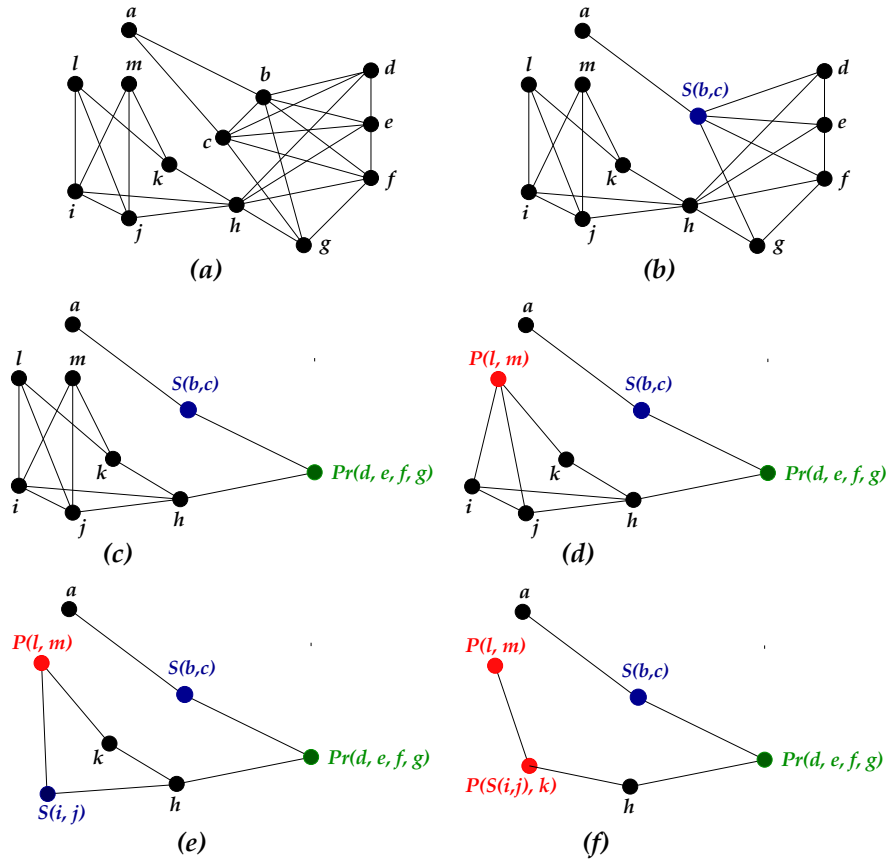


Figure 3: The quotient graph obtained by compacting the strong modules of the graph G. S: series module. P: parallel module. Pr: prime module.

We consider the graph that corresponds to the final quotient graph obtained by compacting

all the strong modules[1] and we call this graph the prime graph. Each vertex of this graph is a maximum strong module which may contain other nested strong modules.

Modular decomposition has been extensively studied by many authors [18, 20, 24, 38, 46]. Several algorithms that compute the modular decomposition of a graph are proposed in the literature [25]. All existing algorithms aim to reduce the time complexity and find the simplest way to obtain the modular decomposition of a graph. The first polynomial algorithm is due to Cowan et al. [17] and runs in $O(n^4)$, where n is the number of vertices in the graph. Habib and Maurer [23] proposed an $O(n^3)$ algorithm. Then, Muller and Spinard [40] proposed an incremental $O(n^2)$ algorithm. Independently, Mc-Connell and Spinard [33], Cournier and Habib [2] and Dahlhaus et al. [18] succeeded to obtain a linear algorithm in $O(n + m)$ time, where n is the number of vertices in the graph and m the number of edges. Finally, Habib et al. [13, 25] developed a simpler linear time algorithm in $O(n + m)$. We refer the reader to [24, 46] for a more detailed survey on modular decomposition algorithms and their algorithmic techniques. All existing algorithms focus on computing the modular decomposition tree of a graph. This tree represents how modules are nested in all the graph. So, we adapted the algorithm of [13, 25] to extract the final quotient graph. This modification consists mainly in:

- Finding the adjacency relationships between the maximum strong modules
- Keeping track of the vertex labels because the modular decomposition algorithm doesn't deal with labels.

3 Graph Distance Measure based on the Prime Graph

Given two labeled graphs G_1 and G_2 and their respective prime graphs $\mathcal{P}(G_1)$ and $\mathcal{P}(G_2)$, we aim to take a decision on the similarity between G_1 and G_2 by comparing their prime graphs. There are at least two advantages for such approach. First, similarity is computed on simpler graphs which reduces considerably the number of comparisons for large graphs. Second, the detection of particular structures within the graph, i.e., the modules, may enhance the accuracy of the comparison mainly for classification problems. We first obtain the prime graph by using a modular decomposition algorithm then we use graph probing and star comparison to compute a distance between these graphs. So, computing the similarity between two graphs G_1 and G_2 is mapped to computing the similarity between their corresponding prime graphs $\mathcal{P}(G_1)$ and $\mathcal{P}(G_2)$. We consider simple labeled graphs which do not contain self-loops, multi-edges and edge labels. An undirected attributed graph, denoted by G can be represented by a 3-tuple $G = (V, E, f)$ where V is a finite set of vertices, $E \subseteq V \times V$ is a set of vertex pairs and f is a function assigning labels to vertices. As the vertices of a prime graph are maximum strong modules, we first define the distance between two strong modules. Subsequently, we will define the distance between

[1]Note that, for some graphs, we can compact entirely the graph into a single node. In this case, we stop the compacting process one step before the entire compacting of the graph to avoid obtaining a trivial module.

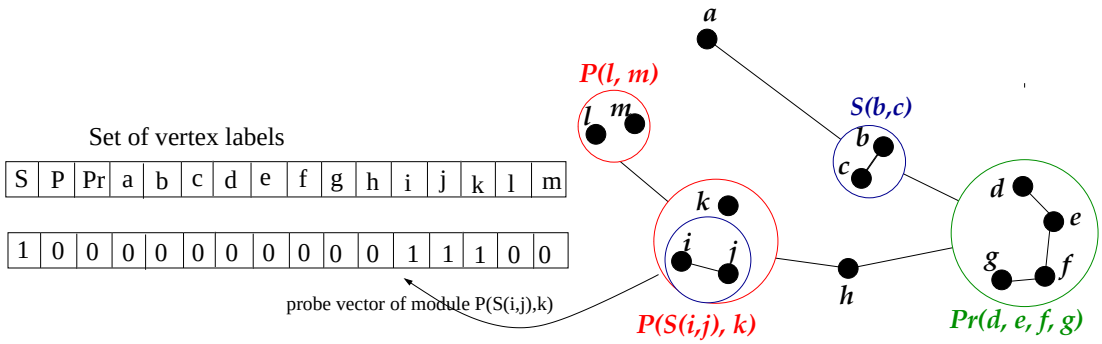


Figure 4: Example of Probe Vector of a module.

stars of strong modules. After that, we will define the mapping distance between two prime graphs based on their star representations. Finally, we will define the mapping distance between two graphs based on the distance between their prime graphs.

As pointed-out in the previous section, we focus on maximum strong modules that ensure the uniqueness of the prime graph. The structure inside a maximum strong module is encoded by the type of the module and also the types of inner strong modules that are contained in it. The type of a module is represented by a vertex labeled **S** (for a series module), **P** (for a parallel module) and **Pr** for (for a prime module). So, to describe a module, we use a label-oriented representation based on a probe vector. This means that we mainly represent vertices rather than edges because the structure inside the modules, i.e. the adjacency relationships, is already encoded in the labels S, P and Pr which characterize the modules.

Definition 5 (***module probe vector***)**.** *Let M be a module in the graph G and let $\mathcal{A} = \{l_S, l_P, l_{Pr}\} \bigcup \{l_0, l_1, l_2, \ldots, l_\alpha\}$ denote an ordered finite set of vertex labels of $\mathcal{P}(G)$. M can be described with a numerical vector of non negative integers $V_M = (a_S, a_P, a_{Pr}, a_0, a_1, a_2, \ldots, a_\alpha)$ such that M contains exactly a_S series modules , a_P parallel modules , a_{Pr} prime modules and a_i vertices labeled l_i.*

Figure 4 shows an example of a probe vector for one of the modules of our graph example (i.e., graph of Figure 2).

We compute the distance between two probe vectors as follows:

Definition 6 (***probe vector distance***)**.** *Given two probe vectors V_1 and V_2, the distance between V_1 and V_2 is given by:*
$\gamma(V_1, V_2) = max(\|V_1\|_1, \|V_2\|_1) - \lfloor \frac{\|V_1+V_2\|_1 - \|V_1-V_2\|_1}{2} \rfloor$

$\gamma(V_1, V_2)$ computes the number of edit operations needed to transform V_1 into V_2 by the addition and the suppression of labels or by modifying labels (i.e, relabeling). It is equivalent to $\mathfrak{M}(\mathfrak{L}_1, \mathfrak{L}_2)$ in Definition 3.

According to this notation, a module structure which is encoded in the type of the module, can be defined as follows:

Definition 7 (***module structure***)**.** *A Module structure T_M can be represented by a 4-tuple $T_M = (r, f, \mathcal{A}, V_M)$ where r is the type of the module, $\mathcal{A}$ is the set of labels, f a labeling function and V_M the probe vector of the module.*

Then, we define the distance between two module structures as follows:

Definition 8 (***module distance***)**.** *Given two module structures $T_{M_1} = (r_1, f_1, \mathcal{A}, V_{M_1})$ and $T_{M_2} = (r_2, f_2, \mathcal{A}, V_{M_2})$ then the distance between T_{M_1} and T_{M_2} is:*

$$d(T_{M_1}, T_{M_2}) = \omega(r_1, r_2) + \gamma(V_{M_1}, V_{M_2}).$$

where

$$\omega(r_1, r_2) = \begin{cases} 0 & \text{if } r_1 = r_2, \\ 1 & \text{otherwise.} \end{cases}$$

$\gamma(V_{M_1}, V_{M_2})$ is given by Definition 6.

Every vertex v_i in a prime graph $\mathcal{P}(G)$ has a corresponding star of modules $S_i^{\mathcal{P}(G)}$ defined as $S_i^{\mathcal{P}(G)} = (r_{S_i}, L_{S_i}, f_i)$ where r_{S_i} is the module at the root of the star $S_i^{\mathcal{P}(G)}$, L_{S_i} is the set of modules of the leaves of the star and f_i is a labeling function. For a prime graph containing k vertices, we have k stars of modules. Accordingly, a prime graph $\mathcal{P}(G)$ can be mapped to a multiset of stars of modules denoted $SM(\mathcal{P}(G))$. The distance between two stars of modules is defined as follows:

Definition 9 (***star of modules distance***)**.** *Given two stars of modules $S_1 = (r_{S_1}, L_{S_1}, f_1)$ and $S_2 = (r_{S_2}, L_{S_2}, f_2)$ then the distance between S_1 and S_2 is:*

$$d_{S_M}(S_1, S_2) = d(r_{S_1}, r_{S_2}) + \psi(L_{S_1}, L_{S_2}) + |\Delta(S_1) - \Delta(S_2)|.$$

where $\psi(L_{S_1}, L_{S_2}) = \gamma(V_{S_1}, V_{S_2})$ with V_{S_i} a probe vector associated to L_{S_i} and given by:
$V_{S_i} = \sum_{M_t \in L_{S_i}} V_{M_t}$.
$d(r_{S_1}, r_{S_2})$ is calculated with Definition 8.

Definition 10 (***graph distance***)**.** *Given two prime graphs $\mathcal{P}(G_1)$ and $\mathcal{P}(G_2)$ of two graphs G_1 and G_2 with their multisets of star decompositions $SM(\mathcal{P}(G_1)) = \{S_1, S_2, \ldots, S_{|V(\mathcal{P}(G_1))|}\}$ and $SM(\mathcal{P}(G_2)) = \{S'_1, S'_2, \ldots, S_{|V(\mathcal{P}(G_2)|}\}$ respectively, then the distance between G_1 and G_2 is:*

$$Pr\_Dist(G_1, G_2) = \min_h \sum_{S_i \in SM(\mathcal{P}(G_1)), h(S_i) \in SM(\mathcal{P}(G_2))} d_{S_M}(S_i, h(S_i)).$$

where
$h : SM(\mathcal{P}(G_1)) \to SM(\mathcal{P}(G_2))$ is a bijection.
$d_{S_M}(S_i, h(S_i))$ is calculated with Definition 9.

Computing this distance is then equivalent to solve the assignment problem in the square matrix $max(|\mathcal{P}(G_1)|, |\mathcal{P}(G_2)|) \times max(|\mathcal{P}(G_1)|, |\mathcal{P}(G_2)|)$ in which each element represents the distance between the i^{th} star in $SM(\mathcal{P}(G_1))$ and j^{th} star in $SM(\mathcal{P}(G_2))$. To do so, we use the Hungarian algorithm [30] to obtain the minimum cost in $O(n^3)$ time. Note that, in the case where $|\mathcal{P}(G_1)| \neq |\mathcal{P}(G_2)|$, the square matrix is obtained by adding empty stars.

4 Complexity Analysis and Discussion

In the remaining sections, we will denote by Ex_Dist the distance obtained by the exact graph edit distance computation based on the A^* algorithm [26] and by St_Dist the distance obtained by the approximate graph distance algorithm based on star assignment [63]. The proposed distance, denoted Pr_Dist is mainly compared to Ex_Dist and St_Dist.

The proposed approach involves three steps:

1. Building the prime graph: this needs $O(n + m)$ time where n is the number of vertices in the graph and m is the number of its edges. We use the modular decomposition algorithm described in [13, 25] and modified to extract the adjacency relationships between the maximal strong modules from the tree generated by the algorithm.

2. Computing the distance between each pair of stars in the obtained prime graphs: we need therefore $O(k^3)$ time steps where k is the number of vertices in the largest prime graph, i.e., the number of maximal strong modules in the largest prime graph.

3. Solve the assignment problem by using the Hungarian algorithm [30] to obtain the minimum cost in $O(k^3)$ time.

So, our algorithm runs in polynomial time and has a complexity of $O(k^3 + n + m)$ where k is the number of vertices in the largest prime graph. In the worst case $k = n$ which means that the graph is not decomposable. In this case the algorithm is equivalent to the algorithm of [63] as stated by the following Lemma.

Lemma 1. *Let G_1 and G_2 be undirected attributed graphs. If G_1 and G_2 are not decomposable then $Pr\_Dist(G_1, G_2) = St\_Dist(G_1, G_2)$.*

Proof. If G_1 and G_2 are not decomposable, we have $G_1 = \mathcal{P}(G_1)$ and $G_2 = \mathcal{P}(G_2)$. So, it suffices to show that for two stars S_1 and S_2 of G_1 and G_2 respectively, we have : $d_{S_M}(S_1, S_2) = \lambda(S_1, S_2)$. λ is given by Definition 3.
$d_{S_M}(S_1, S_2) = d(r_{S_1}, r_{S_2}) + \psi(L_{S_1}, L_{S_2}) + |\Delta(S_1) - \Delta(S_2)|$. As there are no modules, $r_{S_1} = r_1$ and $r_{S_2} = r_2$ with r_1 and r_2 the vertices at the roots of S_1 and S_2 respectively. So, $d(r_{S_1}, r_{S_2}) = \omega(r_1, r_2) = T(r_1, r_2)$.
$\psi(L_{S_1}, L_{S_2}) = \gamma(V_{S_1}, V_{S_2})$ where V_{S_1} gives the number of occurrences of each label in the leaves

of S_1 and V_{S_2} gives the number of occurrences of each label in the leaves of S_2. So, $\|V_{S_1}\|_1 = |\mathfrak{L}_1|$, $\|V_{S_2}\|_1 = |\mathfrak{L}_2|$ and $max\{\|V_{S_1}\|_1, \|V_{S_2}\|_1\} - \lfloor \frac{\|V_{S_1}+V_{S_2}\|_1 - \|V_{S_1}-V_{S_2}\|_1}{2} \rfloor$ has the same result as $\mathfrak{M}(\mathfrak{L}_1, \mathfrak{L}_2)$. Consequently, $d_{S_M}(S_1, S_2) = \lambda(S_1, S_2)$. □

Hence, when the two graphs are not decomposable Pr_Dist behaves similarly to St_Dist when compared to Ex_Dist. The comparison between St_Dist and Ex_Dist is given by:

Theorem 1. *[63]* $St\_Dist(G_1, G_2) \leq max\{4, max\{\Delta(G_1), \Delta(G_2)\} + 1\} Ex\_Dist(G_1, G_2)$

When at least one of the two graphs is decomposable, we have the following relation between Pr_Dist, St_Dist and Ex_Dist:

Theorem 2. *For any graphs G_1 and G_2, the following properties hold:*

1. $Pr\_Dist(G_1, G_2) \geq St\_Dist(G_1, G_2) \geq Ex\_Dist(G_1, G_2)$
2. $Pr\_Dist(G_1, G_2) \leq (5 + 3\, max\{\Delta(G_1), \Delta(G_2)\}) Ex\_Dist(G_1, G_2)$

Proof. We follow the same reasoning as in [63]. Let $P = (p_1, p_2, \cdots, p_k)$ be a set of edit operations transforming G_1 to G_2 in Ex_Dist. This means that there is a sequence of graphs $G_1 = h_0 \rightarrow h_1 \rightarrow \cdots \rightarrow h_k = G_2$, where $h_{i-1} \rightarrow h_i$ indicates that h_i is obtained by applying edit operation p_i on h_{i-1} for $1 \leq i \leq k$. Assume there are k_1 edge insertion/deletion operations, k_2 vertex insertion/deletion operations and k_3 vertex relabeling operations in P with $k_1 + k_2 + k_3 = k$. In the following, we will analyze each kind of edit operations in detail.

- **Vertex Insertion/Deletion**: In the case of vertex insertion, since the newly inserted vertex v_i has no edges, $Pr\_Dist(h_m, h_{m+1}) = St\_Dist(h_m, h_{m+1}) = 1$. The same result is obtained in case of deleting one isolated vertex.
- **Edge Insertion/Deletion**: For St_Dist, inserting an edge between two vertices v_i and v_j in the graph h_m affects the two stars rooted at v_i and v_j respectively. For both stars, a new vertex and a new edge are inserted. So, St_Dist induces 4 edit operations [63].
  For Pr_Dist, inserting an edge between two vertices v_i and v_j in the graph h_m may induce one of the following modifications of the stars and module structures:
  - Modifying two simple stars (i.e., no module is affected): this case is similar to St_Dist and induces 4 edit operations.
  - Creating a new module or splitting up an existing module (see Figure 5): assume there is a vertex v_t in h_m such that when inserting an edge between v_i and v_j, v_i obtains the same neighboring hood as vertex v_t. The two vertices v_i and v_t form a new module M in $\mathcal{P}(h_{m+1})$. Each star in $\mathcal{P}(h_m)$ that has v_i and v_t in its leaves has now M as a leaf in $\mathcal{P}(h_{m+1})$. This means that the number of labels in the leaves of each star increases by 1 (i.e, the label that corresponds to the type of the module) but the degree of the star decreases by 1. The star rooted at v_j has 2 new labels in its leafs (v_i and the

type of the module, i.e., 2 operations). Also, the star routed at v_i and the star rooted at v_t are replaced by one star rooted at M that has 3 labels in the root. This is equivalent to add 2 labels to the star rooted at v_t (i.e., 2 operations) and to suppress the star rooted at v_i (i.e., $1 + 2\,deg(v_i)$). So Pr_Dist induces $2 + 2 + 1 + 3\,deg(v_i)$ edit operations. The same reasoning can be applied when the inserted edge splits up an existing module.

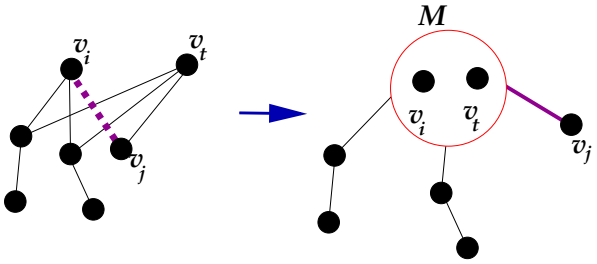


Figure 5: Edge insertion inducing a module creation.

– Modifying an existing module (see Figure 6): Let M be a module in $\mathcal{P}(h_m)$ such that when inserting an edge between v_i and v_j, v_i obtains the same neighboring hood as a vertex v_t of M. Vertex v_i joins the module M. In this case, the star rooted at M has an additional label in the root. The stars that have v_i as a leaf (so have also M as a leaf) decrease their degree by 1 (i.e., $deg(v_i)$ edit operations) and we have one star in less, the star rooted at v_i ((i.e., $1 + 2\,deg(v_i)$). This gives $Pr\_Dist(h_m, h_{m+1}) = 1 + 1\,deg(v_i) + 1 + 2\,deg(v_i)$. So, $Pr\_Dist(h_m, h_{m+1}) = 2 + 3\,deg(v_i)$ in this case. The same reasoning can be applied when the inserted edge retrieves a vertex from an existing module.

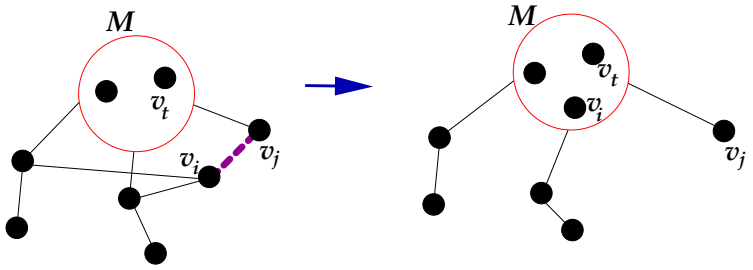


Figure 6: Edge insertion inducing a module modification.

Note that, in the case of modification of an existing module, we have $deg(v_i) > 0$. So, in both cases we have $Pr\_Dist(h_m, h_{m+1}) \geq St\_Dist(h_m, h_{m+1})$.

- **Vertex Relabeling**: Assume a vertex v_i's label is changed from ℓ_1 to ℓ_2. In this case, the star containing v_i in its root and also all the stars containing v_i in their leaves are affected.

Therefore, for one vertex relabeling, we have $Pr\_Dist(h_m, h_{m+1}) = St\_Dist(h_m, h_{m+1})$ and both of them induce $1 + deg(v_i)$) edit operations.

Consequently, we have the following inequalities:

- $Pr\_Dist(G_1, G_2) \geq St\_Dist(G_1, G_2) \geq Ex\_Dist(G_1, G_2)$, and
- $$\begin{aligned} Pr\_Dist(G_1, G_2) &\leq max(4, 5 + 2\,deg(v_i), 2 + 3\,deg(v_i))k_1 + k_2 + (1 + deg(v_i))k_3 \\ &\leq (5 + 3\,max\{\Delta(G_1), \Delta(G_2)\})\,k_1 + k_2 + (max\{\Delta(G_1), \Delta(G_2)\} + 1)\,k_3 \\ &\leq (5 + 3\,max\{\Delta(G_1), \Delta(G_2)\})\,(k1 + k2 + k3) \\ &\leq (5 + 3\,max\{\Delta(G_1), \Delta(G_2)\})Ex\_Dist(G_1, G_2) \end{aligned}$$

□

Theorem 3. *For any graphs G_1, G_2 and G_3, the following properties hold:*

1. $Pr\_Dist(G_1, G_2) \geq 0$
2. $Pr\_Dist(G_1, G_2) = Pr\_Dist(G_2, G_1)$
3. $Pr\_Dist(G_1, G_2) \leq Pr\_Dist(G_1, G_3) + Pr\_Dist(G_2, G_3)$

Proof. Properties 1 and 2 follow directly from the uniqueness of the prime graphs obtained by the relations of adjacency between maximum strong modules and from Definitions 5-10. We focus here on the proof of the triangle inequality. The distance between two graphs is the minimum cost between stars given by the assignment algorithm. This algorithm preserves the triangle inequality, so we have to prove the triangle inequality between stars. Let S_1, S_2 and S_3 three stars of modules from $\mathcal{P}(G_1)$, $\mathcal{P}(G_2)$ and $\mathcal{P}(G_3)$ respectively, we need to show that:

$$d_{S_M}(S_1, S_2) \leq d_{S_M}(S_1, S_3) + d_{S_M}(S_3, S_2)$$

with $d_{S_M}(S_i, S_j) = d(r_{S_i}, r_{S_j}) + \psi(L_{S_i}, L_{S_j}) + |\Delta(S_i) - \Delta(S_j)|$ according to Definition 9. In this equation, $d(r_{S_i}, r_{S_j})$ is equal to 0 or 1, $|\Delta(S_i) - \Delta(S_j)|$ is the difference of degrees of the two stars and $\psi(L_{S_i}, L_{S_j})$ is the number of edit operations that are needed to obtain the leaves of S_i from the leaves of S_j and vice versa. As, we consider vector of probes to represent the leaves, we consider only edit operations on labels. Property 3 in Theorem 3 is equivalent to the following inequality:

$$d(r_{S_1}, r_{S_2}) + \psi(L_{S_1}, L_{S_2}) + |\Delta(S_1) - \Delta(S_2)| \leq d(r_{S_1}, r_{S_3}) + \psi(L_{S_1}, L_{S_3}) + |\Delta(S_1) - \Delta(S_3)| +$$

$$d(r_{S_3}, r_{S_2}) + \psi(L_{S_3}, L_{S_2}) + |\Delta(S_3) - \Delta(S_2)| \quad (1)$$

For the notational convenience, let $r_{12} = d(r_{S_1}, r_{S_2})$, $r_{13} = d(r_{S_1}, r_{S_3})$, $r_{32} = d(r_{S_3}, r_{S_2})$, $a_{12} = |\Delta(S_1) - \Delta(S_2)|$, $a_{13} = |\Delta(S_1) - \Delta(S_3)|$ and $a_{32} = |\Delta(S_3) - \Delta(S_2)|$. We also denote by m_{11}, m_{22}, m_{33}, m_{12}, m_{13}, m_{23} and m_{123} the different pieces of overlapping between the labels of L_{S_1}, L_{S_2} and L_{S_3} as depicted in Figure 7.

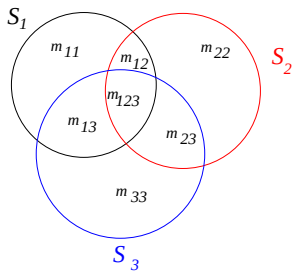


Figure 7: Overlapping between labels of L_{S_1}, L_{S_2} and L_{S_3}

Equation 1 is equivalent to:

$$r_{12}+m_{11}+m_{22}+m_{13}+m_{23}+a_{12} \leq r_{13}+m_{11}+m_{33}+m_{12}+m_{23}+a_{13}+r_{32}+m_{22}+m_{33}+m_{13}+m_{12}+a_{32} \quad (2)$$

which can be simplified to:

$$r_{12} + a_{12} \leq r_{13} + r_{32} + a_{13} + a_{32} + 2(m_{33} + m_{12}) \quad (3)$$

We consider six cases according to the degrees of the stars:

- Case 1: $\Delta(S_1) \geq \Delta(S_2) \geq \Delta(S_3)$. Here, we have $a_{13} \geq a_{13} \geq 0$ and $a_{13} \geq a_{23} \geq 0$. Then, according to the value of r_{12}, we have:
  - $r_{12} = 0$, here Equation 3 holds whatever is in the value of its right part.
  - $r_{12} = 1$, this means that $r_{13} = 1$ or $r_{23} = 1$ and consequently Equation 3 holds.
- Case 2: $\Delta(S_1) \geq \Delta(S_3) \geq \Delta(S_2)$. Here, a_{12}, a_{13} and a_{23} can be computed by:
  $a_{12} = m_{11} + m_{13} - m_{22} - m_{23}$
  $a_{13} = m_{11} + m_{12} - m_{33} - m_{23}$
  $a_{23} = m_{33} + m_{23} - m_{22} - m_{12}$
  Equation 3 is simplified to: $r_{12} \leq r_{13} + r_{32} + 2(m_{33} + m_{12})$ which holds true.

The remaining four cases $\Delta(S_2) \geq \Delta(S_1) \geq \Delta(S_3)$, $\Delta(S_2) \geq \Delta(S_3) \geq \Delta(S_1)$, $\Delta(S_3) \geq \Delta(S_1) \geq \Delta(S_2)$ and $\Delta(S_3) \geq \Delta(S_2) \geq \Delta(S_1)$ can be shown similarly. □

Note that the prime graph based distance does not respect the uniqueness property and two non isomorphic graphs G_1 and G_2 may verify $Pr\_Dist(G_1, G_2) = 0$. So, the Prime graph based distance for labeled undirected graphs is a pseudo-metric.

5 Evaluation

In this section, we evaluate how much our approach approximates the similarity between two graphs and to which extent it can be used for pattern recognition and classification tasks. We also evaluate its execution time performance. For modular decomposition, we use the algorithm proposed in [13, 25] which is linear time. It generates the modular decomposition of a graph of n vertices and m edges in $O(n+m)$ time.
All experiments were conducted on a 2.80 GHz $Intel(R)\ Core(TM)\ i7-2640M$ 64 bits laptop with 8 GB main memory running on Windows 7. All programs were implemented in C^{++}. For comparison, we use two reference distances. The optimal A^* based algorithm of exact graph edit distance computation [26] described in Section 1 and the approximate graph edit distance algorithm proposed in [63] and also described in Section 1.

5.1 Datasets

We investigate the precision of our graph distance over several real datasets from the IAM graph database repository[2] for graph-based pattern recognition and machine learning [50] and the RI database of biochemical data[3] [4]. In Table 2 a summary of these graph data. Besides the size of each dataset, its maximum number of vertices and edges and its average number of vertices and edges, we also give the average and maximum *reduction rates* of the dataset. Given a graph G and its prime graph $\mathcal{P}(G)$, the reduction rate RR of G is given by: $RR(G) = \frac{|V(G)|-|V(\mathcal{P}(G))|}{|V(G)|} \cdot 100\%$. It compares the number of vertices in $\mathcal{P}(G)$ in respect to G.

Table 2: Graph Dataset Characteristics
tr: size of the training set. **va**: size of the validation set. **te**: size of the test set. #classes: number of classes. $avg|V|$: average number of vertices. $avg|E|$: average number of edges. $max|V|$: maximum number of vertices. $max|E|$: maximum number of edges. $avgRR(\%)$: average reduction rate of the dataset and $maxRR(\%)$ its maximum reduction rate.

Dataset	**size(tr,va,te)**	**#classes**	$avg\|V\|$	$avg\|E\|$	$max\|V\|$	$max\|E\|$	$avgRR(\%)$	$maxRR(\%)$
$AIDS$	250,250,1500	2	15.69	16.19	95	103	53.219	94.199
$Mutagenecity$	1500,500,2337	2	30.32	30.78	417	112	20.833	97.559
$GREC$	286,286,528	22	11.51	11.93	24	29	32.267	91.669
$Protein$	200,200,200	6	32.63	62.14	126	149	16.860	95.349
$Letter$	6750	15	4.67	3.61	9	9	27.52	75
PPI	10	-	7828.9	107134.8	12578	332458	3.41	6.07
$PDBS$	30	-	7448.2	5628.9	33067	30773	63.28	88.52

1. The AIDS database (IAM database): The AIDS dataset consists of graphs representing molecular compounds. This dataset consists of two classes (active, inactive), which represent molecules with activity against HIV or not. Graphs are constructed from the AIDS Antiviral Screen Database of Active Compounds [50]. The molecules are converted into graphs by representing atoms as vertices and the covalent bonds as edges [50]. Vertices are labeled with the number of the corresponding chemical symbol and edges[4] by the valence

[2] www.iam.unibe.ch/fki/databases/iam-graph-database
[3] http://ferrolab.dmi.unict.it/ri/ri.html#description
[4] Note that we have not used edge labels for all the datasets.

of the linkage. The resulting graphs are small and sparse. We use a training set and a validation set of size 250 each, and a test set of size 1500. Thus, there are 2000 elements totally (1600 inactive elements and 400 active elements).

2. The Protein database(IAM database): The protein dataset consists of graphs representing proteins originally used in [6]. The proteins database consists of six classes, which represent proteins out of the six enzyme commission top level hierarchy (EC classes). The proteins are converted into graphs [50] by representing the secondary structure elements of a protein with vertices and edges of an attributed graph. Vertices are labeled with their amino acid sequence (e.g. TFKEVVRLT). Every vertex is connected with an edge to its three nearest neighbors in space. Edges are labeled with their type and the distance they represent in angstroms. There are 600 proteins totally, 100 per class. We use a training, validation and test sets of equal size (200).

3. The Mutagenicity database (IAM database): Mutagenicity is a kind of chemical compounds. The molecules were converted into graphs [50] by representing atoms as vertices and the covalent bonds as edges. Vertices are labeled with the number of the corresponding chemical symbol and edges by the valence of the linkage. The mutagenicity dataset is divided into two classes mutagen and nonmutagen. The dataset contains 4337 elements totally (2401 mutagen elements and 1936 nonmutagen elements). We use a training set, a validation set and a test set of equal size 300.

4. GREC database (IAM database): The GREC dataset consists of graphs representing symbols from architectural and electronic drawings. Graphs are extracted from images by tracing the lines from end to end and detecting intersections as well as corners [50]. Ending points, corners, intersections and circles are represented by vertices and labeled with a two-dimensional attribute giving their position. The vertices are connected by undirected edges which are labeled as "line" or "arc". An additional attribute specifies the angle with respect to the horizontal direction or the diameter in case of arcs [50]. The obtained graphs are uniformly distributed over 22 classes. The resulting set is split into a training and a validation set of size 286 each, and a test set of size 528.

5. Letter database (IAM database): This graph dataset contains graphs that represent distorted letter drawings [50]. Only the 15 capital letters of the Roman alphabet that consist of straight lines (A, E, F, H, I, K, L, M, N, T, V, W, X, Y, Z) are considered. For each class, a prototype line drawing is manually constructed. These prototype drawings are then converted into prototype graphs by representing lines by undirected edges and ending points of lines by nodes. Each node is labeled with a two-dimensional attribute giving its position relative to a reference coordinate system [50]. Edges are unlabeled. The graphs are uniformly distributed over the 15 classes. Distortions are applied on the prototype graphs with three different levels of strength, viz. low, medium and high. Hence, this dataset contains 3 subsets: HIGH, MED and LOW with 2250 graphs in each set.

6. The PPI database (RI database): This dataset contains graphs describing the known and predicted protein interactions. The graphs describe the following organisms: Mus musculus, Saccaromyces cerevisiae, Caenorhabditis elegans, Drosophila melanogaster, Takifugu rubipres, Danio rerio, Xenopus tropicalis, Bos taurus,Rattus norvegicus, and Homo sapiens. They are large graphs. The original version of the dataset have unique vertex labels (protein IDs) [57]. We use the RI version which was randomly relabeled with 2048 labels. Labels are assigned using a uniform distribution [4].

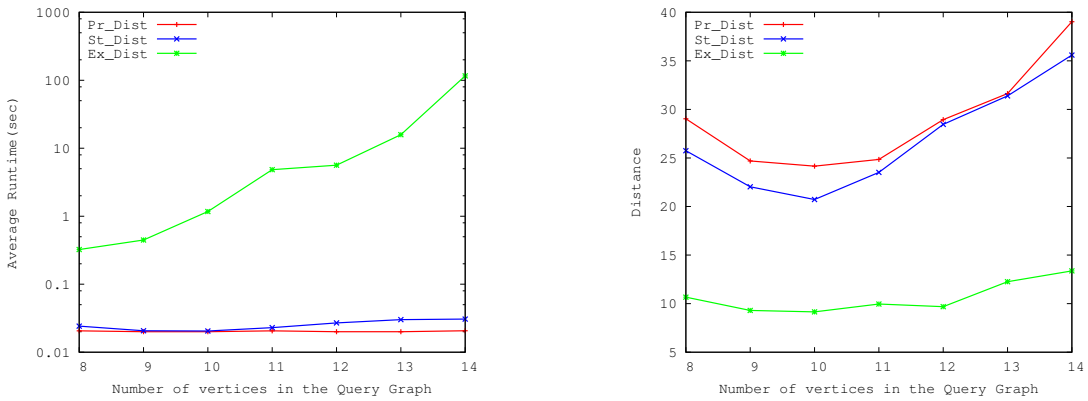
7. The PDBS database (RI Database): This dataset contains 30 graphs with data from DNA, RNA, and proteins having up to 33067 vertices. The dataset mostly contains very large graphs [4].

5.2 Results and Discussion

5.2.1 Comparison with the Exact Graph Edit Distance Algorithm

We first conducted experiments to compare our algorithm to the exact graph edit distance computation based on the A^* algorithm [26], denoted by Ex_Dist. We also compared our algorithm to the approximate graph edit distance computation algorithm based on subgraph assignment [28,47,49,63]. We implemented the solution of [63] denoted here by St_Dist mainly because it does not use edge labels and consequently can be easily compared to the proposed solution. We consider two metrics: runtime and also how the proposed distance behaves with respect to the exact edit distance computation. For both metrics, we used the AIDS datasets and considered only small graphs to avoid the time complexity related to the exact computation of the graph edit distance. From the 2000 graphs of the AIDS dataset, we randomly selected 10 graphs each of which contains 10 vertices to form a target graph database. We also constructed 7 query groups: Q_8,...,Q_{14} each of which contains 10 graphs. All the graphs of the same query group Q_i have the same number of vertices. The number of vertices in each graph of a query group Q_i is i. This means that the number of vertices of the query graphs varies from 8 to 14. Figure 8 (a) shows the average runtime performance of the three distances. The X-axis shows the number of vertices contained in the query graph and the Y-axis the average runtime, in log scale, obtained over the query group of the corresponding graph size when compared to the target graph database. This figure shows clearly that St_Dist and Pr_Dist are faster than Ex_Dist. We can also see that Pr_Dist achieves a little bit better than St_Dist.

Then, we focused on how Pr_Dist behaves compared to the exact edit distance Ex_Dist and to its approximation given by St_Dist. Figure 8 (b) shows the average value of the three distances computed over each query group. We use the same AIDS dataset as for the evaluation of the runtime. Based on this figure, we can see that $Ex\_Dist \leq St\_Dist \leq Pr\_Dist$ which confirm our theoritical study. We can also see that both Pr_Dist and St_Dist follow the curve of Ex_Dist in general.



(a) Runtime Vs number of vertices

(b) Distance Vs number of vertices

Figure 8: Comparison with Ex_Dist

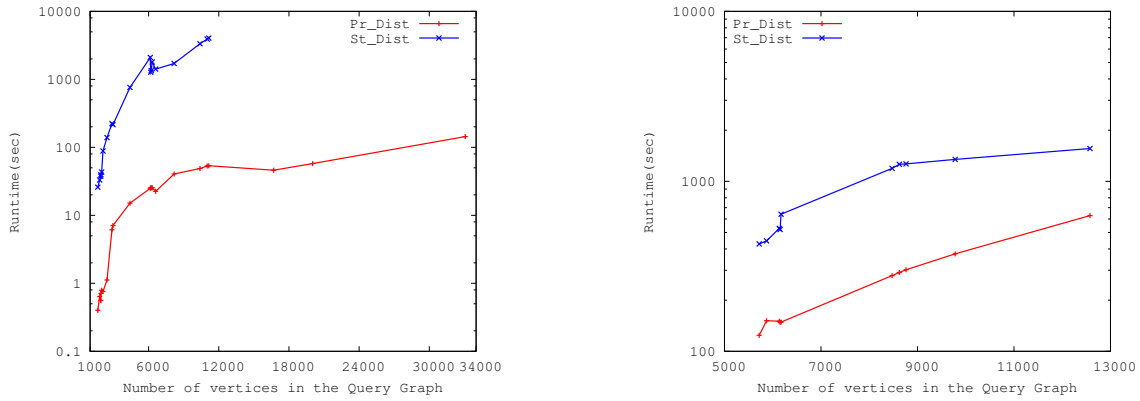
5.2.2 Scalability over Large Graphs

We evaluated the scalability of our approach in terms of the order of the graphs, i.e., the number of vertices of the graphs. For this, we used two datasets: PDBS and PPI. Both datasets contain large graphs with a significant difference in the reduction rate : 63.28% for PDBS and 3.41% for PPI. For each dataset, we used all the the graphs that have a number of vertices lower than 12000 as the graph database and we varied the size of the graph query. Figure 9 shows the runtime for calculating Pr_Dist and St_Dist between each query graph and the graph database for the two datasets. The X-axis shows the number of vertices contained in the query graph and the Y-axis the average runtime, in log scale, obtained over the target database. From this figure we can see that Pr_Dist faster than St_Dist in the two datasets and that this difference of performance is proportional to the compression rate. This is mainly due to the reduction but also to the fact that St_Dist uses a time consuming algorithm to compute intersections between multisets. This is avoided by Pr_Dist which is based on probes. We can also see that the runtime performance of Pr_Dist is more important with the PDBS dataset than with the PPI dataset this is due to the difference of reduction rate between the two datasets.

It is also worth noting that memory requirement is also important for large graphs. For the PDBS dataset where some graphs exceed 33000 vertices St_Dist generates an *Out of memory exception* with graphs larger than 12000 vertices while Pr_Dist processes all the graphs. In fact, with an average reduction rate of 63.28%, the largest graph of the PDBS dataset, i.e. 33067 vertices, has a prime graph of at most 12142 vertices. We note also that the two algorithms use the same data structures for graphs and use the same implementation of the Hungarian algorithm and that the *Out of memory exception* generated by St_Dist is due to the Hungarian part of the implementation.

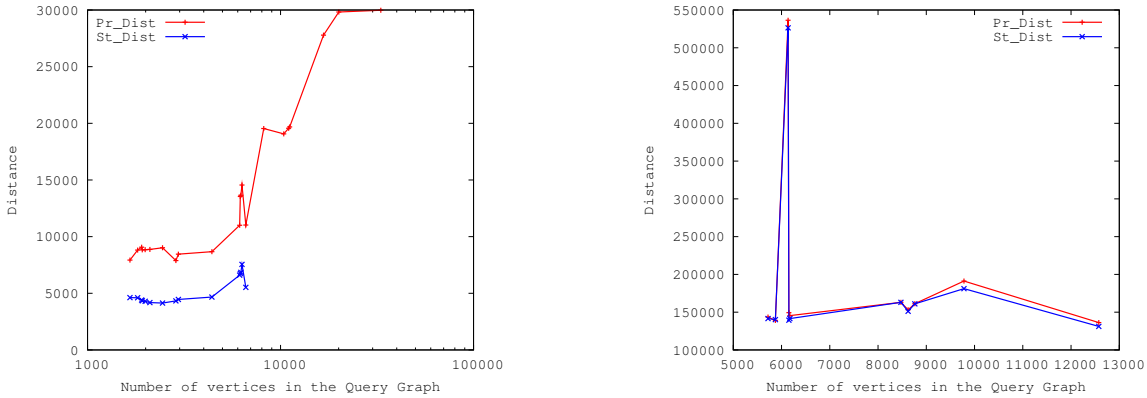
Figure 10 shows the computed distance for Pr_Dist and St_Dist between each query graph and the graph database for the two datasets. The X-axis shows the number of vertices contained

in the query graph (in log scale for PDBS dataset) and the Y-axis the average distance obtained over the target database. From this figure we can see that Pr_Dist is an upper bound for St_Dist. For the PPI dataset that has a small reduction rate the two distances are similar. This confirm the result stated by Lemma 1.



(a) PDBS dataset: $RR = 63.28\%$. (b) PPI dataset: $RR = 3.41\%$.

Figure 9: Scalability over Large Graphs: Runtime Performance



(a) PDBS dataset: $RR = 63.28\%$. (b) PPI dataset: $RR = 3.41\%$.

Figure 10: Scalability over Large Graphs: Computed Distance

5.2.3 Classification

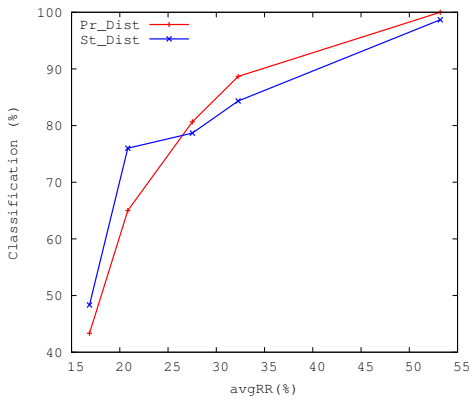
We use the NN classifier to evaluate the quality of the proposed distance and we compared it with St_Dist. Table 3 reports the average classification results obtained for each dataset over the three subsets: Test, train and valid. From these results, we can see that Pr_Dist obtains a better classification for the AIDS, GREC and Letter databases and not for Mutagenecity and Protein databases. It seems that more the dataset is reducible more Pr_Dist is accurate. This

is clearly illustrated in Figure 11. In this figure, the X-axis shows the average reduction rate of the datasets and the Y-axis the average classification obtained over the dataset. Figure 11(a) shows clearly that the classification obtained by Pr_Dist increases with the average reduction rate. It also shows that beyond 27% of reduction Pr_Dist achieves better than St_Dist. To explain these results, we also investigated the impact of the distribution of the reduction over the dataset. In fact, we remarked that when there is a large difference between the reduction rates of the compared graphs, the computed distance tend to be biased by the important number of empty stars needed by the assignment computation. In fact, even if the initial graphs have the same number of vertices, their prime graphs may have very different number of vertices. So, the distribution of the reduction over the dataset is to be considered to explain in which cases Pr_Dist behaves better than St_Dist and in which cases it does not. We considered the difference between the maximum reduction rate and the average reduction rate as a measure of how homogenous is the reduction over a dataset and we plot the results obtained for classification in function of this difference. Figure 11 (b) reports the obtained results. In this figure, the X-axis shows the difference between the maximum and the average reduction rates of the datasets and the Y-axis the average classification obtained over the dataset. It appears from Figure 11(b) that more the reduction is homogenous over the dataset more Pr_Dist is accurate. Figure 11(b) also shows that beyond a difference of reduction of 52%, Pr_Dist looses in accuracy.

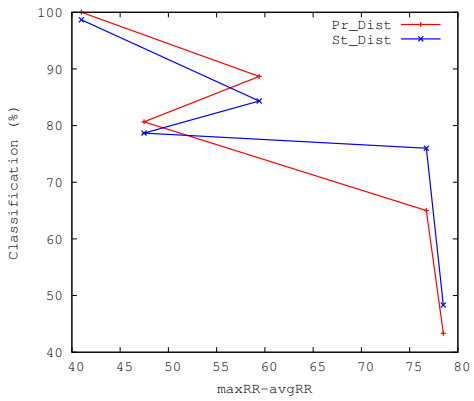
Over all these results, we can conclude that Pr_Dist takes advantage from detecting particular structures in the compared graphs and enhance the accuracy of the classification. However, when the compared graphs are not homogenously reduced, the detection of these particular substructures is not benefic for the distance and Pr_Dist looses in accuracy.

Table 3: Classification

Dataset	St_Dist	Pr_Dist
$AIDS$	98.67	100
$Mutagenecity$	76	65
$GREC$	84	88.67
$Protein$	48.33	43.33
$Letter$	78.67	80.67



(a) Classification Vs Average Reduction Rate.



(b)Classification Vs MaxRR-AvgRR.

Figure 11: Impact of the reduction rate on Classification.

6 Conclusion

In this paper, we introduced an approximate approach for the similarity measure of large graphs which is based on prime graphs. The key idea of the proposed method is to perform the comparison on simpler graphs without losing the structural information of original graphs in order to enhance the time complexity. We provide an experimental evaluation over small and large graphs. The obtained results show that our approach has an interesting runtime performance and has a good scalability in terms of the number of vertices in the compared graphs. Classification results over several real datasets show that the accuracy of the approach is acceptable. Moreover, we also showed that the proposed similarity takes advantage from detecting particular structures in the compared graphs and enhance the accuracy of the classification. However, this result is obtained when the compared graphs are decomposable, hence the need of thresholding the compression of the compared graphs. According to the experiments, this threshold, given by the average reduction rate, can be fixed to 27%. We also remark that the homogeneity of the reduction over the dataset is an important factor to consider when using this method for classification purposes.

As future work, several extensions and enhancements are possible. A non exhaustive list is:

- Use prime graphs or strong modules to compute footprints that may be used for indexing in large graph databases.
- Use other methods to compare prime graphs. To compute our similarity measure, we coupled the prime graphs with a subdivision into stars and we used probe vectors. However, it may be interesting to investigate other ways to use prime graphs to compute a similarity between graphs. Tree edit distance may provide better performance. Indeed, each vertex of the prime graph is a maximum strong module which may contains other nested strong modules. So, it can be represented by a tree. Internal vertices of the tree gives the type of

the modules nested in the maximum strong module and the leaves are the trivial modules, i.e. the vertices of the original graph. Figure 12 illustrates the representation of the strong modules within the final quotient graph of our graph example (graph used in Figures 2 and 3).

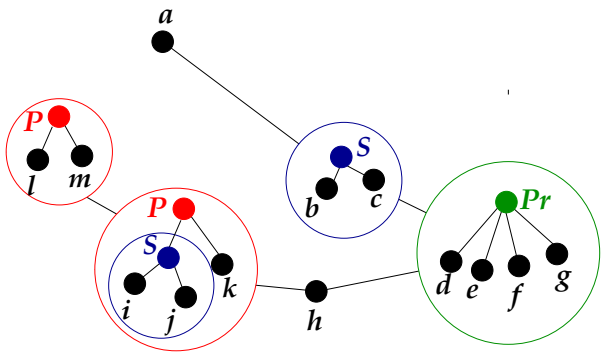


Figure 12: Tree representation of the maximal strong modules within the quotient graph.

- Consider a parallel version of the algorithm to enhance the processing of very large graphs.
- Extend the approach to directed graphs. Note that a modular decomposition algorithm for directed graphs is proposed in [17, 32].
- Extend the approach to edge-labeled graphs. A solution is directly obtained by using a second probe vector for the edge labels of each maximum strong module.

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable comments on earlier drafts of this paper.

References

[1] Sonia Abbas and Hamida Seba. A Module-based Approach for Structural Matching of Process Models. In *The 5th IEEE International Conference on Service Oriented Computing & Applications (SOCA 2012)*, December 2012.

[2] A.Cournier and M.Habib. A new linear algorithm for modular decomposition. *Lecture Notes in Computer Science*, 787:68–84, 1994.

[3] R. Ambauen, S. Fischer, and Horst Bunke. Graph edit distance with node splitting and merging and its application to diatom identification. *Graph Based Representations in Pattern Recognition - GBR*, pages 95–106, 2003.

[4] Vincenzo Bonnici, Rosalba Giugno, Alfredo Pulvirenti, Dennis Shasha, and Alfredo Ferro. A subgraph isomorphism algorithm and its application to biochemical data. *BMC Bioinformatics*, 14(Suppl 7)(S13), 2013.

[5] K. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *5th Int. Conference on Data Mining*, pages 74–81, 2005.

[6] K. Borgwardt, C. Ong, S. Schönauer, S. Vishwanathan, A. Smola, and H.-P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(1):47–56, 2005.

[7] H. Bunke. Error correcting graph matching: On the influence of the underlying cost function. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(9):917–922, 1999.

[8] H. Bunke and G. Allerman. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters - PRL*, 1(4):245–253, 1983.

[9] H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3-4):255–259, 1998.

[10] Horst Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18:689–694, 1997.

[11] Horst Bunke and Bruno T. Messmer. Recent Advances in Graph Matching. *International Journal of Pattern Recognition and Artificial Intelligence*, 11:169–203, 1997.

[12] Horst Bunke and Kaspar Riesen. Recent advances in graph-based pattern recognition with applications in document analysis. *Pattern Recognition*, 44:1057–1067, 2011.

[13] Christian Capelle, Michel Habib, and Fabien De Montgolfier. Graph decompositions and factorizing permutations. *Discrete Mathematics & Theoretical Computer Science - DMTCS*, 5(1):55–70, 2002.

[14] Marco Carcassoni and Edwin R. Hancock. Weighted graph-matching using modal clusters. pages 142–151, 2001.

[15] William J. Christmas, Josef Kittler, and Maria Petrou. Structural matching in computer vision using probabilistic relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence - TPAMI*, 17(8):749–764, 1995.

[16] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty Years of Graph Matching in Pattern Recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18:265–298, 2004.

[17] D.D. Cowan, I.O. James, and R.G. Stanton. Graph decomposition for undirected graphs. *3rd S-E Conf. on Combinatorics, Graph Theory and Computing*, pages 281–290, 1972.

[18] E. Dahlhaus, J. Gustedt, and R. McConnell. Efficient and practical modular decomposition. In *eighth annual ACM-SIAM symposium on Discrete algorithms*, pages 26–35, 1997.

[19] Brian Gallagher. *Matching Structure and Semantics: A Survey on Graph-Based Pattern Matching*. PhD thesis, 1939.

[20] T. Gallai. Transitiv orientierbare graphen. *Acta Mathematica Hungarica*, 18:25–66, 1967.

[21] X. Gao, B. Xiao, D. Tao, and X. Li. A survey of graph edit distance. *Pattern Analysis Applications*, 13:113–129, 2010.

[22] T. Gartner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In Springer, editor, *Annual Conf. Computational Learning Theory*, pages 129–143, 2003.

[23] M. Habib and M.C. Maurer. On the x-joint decomposition for undirected graphs. *Discrete Applied Mathematics*, 3:198–207, 1979.

[24] M. Habib and C. Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010.

[25] Michel Habib, Fabien De Montgolfier, and Christophe Paul. A simple linear-time modular decomposition algorithm for graphs. *Scandinavian Workshop on Algorithm Theory - SWAT*, pages 187–198, 2004.

[26] Peter Hart, Nils Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4:100–107, 1968.

[27] D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, University of California, Santa Cruz, 1999.

[28] S. Jouili and S. Tabbone. Attributed graph matching using local descriptions. In *ACIVS 2009, LNCS 5807*, pages 89–99, 2009.

[29] K. G. Khoo and P. N. Suganthan. Multiple relational graphs mapping using genetic algorithms. pages 727–737, 2001.

[30] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[31] Daniel P. Lopresti and Gordon T. Wilfong. Comparing Semi-Structured Documents via Graph Probing. In *Workshop on Multimedia Information Systems*, pages 41–50, 2001.

[32] Ross M. Mcconnell and Fabien De Montgolfier. Linear-time modular decomposition of directed graphs. *Discrete Applied Mathematics*, 145:198–209, 2005.

[33] R.M. McConnelle and J. Spinard. Linear time modular decomposition and efficient transitive orientation of comparability graphs. In *5th ACM-SIAM Symp*, pages 536–545, 1994.

[34] B. McKay. Practical graph isomorphism. *Congress Numerantium*, 87:30–45, 1981.

[35] B. Messmer. *Efficient Graph Matching Algorithms for Preprocessed Model Graphs*. PhD thesis, University of Bern, Switzerland, 1995.

[36] Bruno T. Messmer and Horst Bunke. A decision tree approach to graph and subgraph isomorphism detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence - TPAMI*, 32(12):1979–1998, 1999.

[37] Alessio Micheli. Neural network for graphs : A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

[38] R.H. Möhring. Algorithmic aspect of the substitution decomposition in optimization over relation, set system and boolean function. *Ann. Operations Research*, 4:195–225, 1985.

[39] R.H. Möhring. Algorithmic aspects of comparability graphs and interval graphs. *I. Rival. Graphs and Order (D. Reidel)*, pages 41–101, 1985.

[40] John H. Muller and Jeremy Spinrad. Incremental modular decomposition. *Journal of The ACM - JACM*, 36(1):1–19, 1989.

[41] Richard Myers, Richard C. Wilson, and Edwin R. Hancock. Bayesian graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence - TPAMI*, 22(6):628–635, 2000.

[42] M. Neuhaus and H. Bunke. A convolution edit kernel for error-tolerant graph matching. In *IEEE international conference on pattern recognition, Hong Kong*, pages 220–223, 2006.

[43] Michel Neuhaus and Horst Bunke. An Error-Tolerant Approximate Matching Algorithm for Attributed Planar Graphs and Its Application to Fingerprint Classification. In *International Workshop on Structural and Syntactic Pattern Recognition*, pages 180–189, 2004.

[44] Michel Neuhaus and Horst Bunke. A Random Walk Kernel Derived from Graph Edit Distance. In *International Workshop on Structural and Syntactic Pattern Recognition*, pages 191–199, 2006.

[45] Michel Neuhaus and Horst Bunke. Automatic learning of cost functions for graph edit distance. *Information Sciences*, 177:239–247, 2007.

[46] Ruzayn Quaddoura and Khalid Mansour. Classical graphs decomposition and their totally 2010 decomposable graphs. *International Journal of Computer Science and Network Security*, 10:1240–1250, 2010.

[47] Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. A graph matching method and a graph matching distance based on subgraph assignments. *Pattern Recognition Letters*, 31:394–406, 2010.

[48] John W. Raymond, Eleanor J. Gardiner, and Peter Willett. RASCAL: Calculation of Graph Similarity using Maximum Common Edge Subgraphs. *The Computer Journal*, 45:631–644, 2002.

[49] K. Riesen and H. Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, 27:950–959, 2009.

[50] Kaspar Riesen and Horst Bunke. *IAM Graph Database Repository for Graph Based Pattern Recognition and Machine Learning*. 2008.

[51] Antonio Robles-kelly and Edwin R. Hancock. Graph Edit Distance from Spectral Seriation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27:365–378, 2005.

[52] Antonio Robles-kelly and Edwin R. Hancock. A riemannian approach to graph embedding. *Pattern Recognition -PR*, 40(3):1042–1056, 2007.

[53] A. Sanfeliu and K.S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics (Part B)*, 13(3):353–363, 1983.

[54] Alberto Sanfeliu, René Alquézar, and Francesc Serratosa. Clustering of attributed graphs and unsupervised synthesis of function-described graphs. volume 2, pages 6022–6025, 2000.

[55] J. P Spinrad. *Efficient Graph Representation*. American Mathematical Society, 2003.

[56] Ponnuthurai N. Suganthan. Structural pattern recognition using genetic algorithms. *Pattern Recognition - PR*, 35(9):1883–1893, 2002.

[57] D. Szklarczyk, A. Franceschini, M. Kuhn, M. Simonovic, A. Roth, P. Minguez, T. Doerks, M. Stark, J. Muller, P. Bork, L. Jensen, and C. Von Mering. The string database in 2011: functional interaction networks of proteins, globally integrated and scored. *Nucleic Acids Res*, 39:D561–D568, 2011.

[58] J. R. Ullmann. An Algorithm for Subgraph Isomorphism. *J. ACM*, 23(1):31–42, January 1976.

[59] Shinji Umeyama. An eigen decomposition approach to wighted graph mathcing problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence - TPAMI*, 10(5):695–703, 1988.

[60] W.D. Wallis, P. Shoubridge, M. Kraetz, and D. Ray. Graph distances using graph union. *Pattern Recognition Letters*, 22(6-7):701 – 704, 2001.

[61] Yanghua Xiao, Hua Dong, Wentao Wu, Momiao Xiong, Wei Wang, and Baile Shi. Structure-based graph distance measures of high degree of precision. *Pattern Recognition*, 41:3547–3561, 2008.

[62] Said Yahiaoui, Mohammed Haddad, Brice Effantin, and Hamamache Kheddouci. Coloring based approach for matching unrooted and/or unordered trees. *Pattern Recognition Letters*, 34(6):686 – 695, 2013.

[63] Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *Proceedings of The Vldb Endowment - PVLDB*, 2(1):25–36, 2009.

[64] Xiang Zhao, Chuan Xiao, Xuemin Lin, Qing Liu, and Wenjie Zhang. A Partition-Based Approach to Structure Similarity Search. *Proceedings of The Vldb Endowment - PVLDB*, 7:169–180, 2014.